I am building a testbed to evaluate the energy efficiency of different neural network training algorithms. My ultimate goal is to use the **k-sparse parity problem** as the "Drosophila" (fruit fly) of algorithmic benchmarking, allowing me to use AI to reinvent learning algorithms and compare their energy efficiency (measured by steps/compute to convergence) against a "default method" (standard Gradient Descent).
Please write a self-contained Python script using PyTorch that implements a baseline toy example for this task. The code must be highly modular so I can easily swap out the default optimizer for experimental AI-generated learning algorithms later.
Please include the following components:

1. Data Generation:

- Create a dataset for the (n, k)-sparse parity task. Set $n=3$ (total bits) and $k=3$ (relevant bits).
- The input x should be uniformly sampled from $\{-1, 1\}^n$.
- The target y should be the product (parity) of a fixed, hidden subset of k indices: $y = \prod_{i \in S} x_i \in \{-1, 1\}$.
- Generate a training set of $N=10$ samples and a fixed test set of 10 samples.

2. Model Definition:

- Implement a 2-layer Multi-Layer Perceptron (MLP) with a ReLU activation.
- The hidden layer should have 1000 neurons.
- The output should be a single scalar.

3. Baseline Optimizer and Loss:

- Use standard Stochastic Gradient Descent (SGD) with a learning rate of 0.1.
- Include weight decay ($\lambda = 0.01$), as this is known to be required for standard neural networks to transition from memorization to generalization (grokking) on this task.
- Use Hinge Loss: $\max(0, 1 - f(x) \cdot y)$.

4. Training Loop & Energy Tracking:

- Train using full batches
- Track both training accuracy and test accuracy at every epoch.
- **Crucial Metric:** Implement an "energy tracker" that counts the total number of optimization steps (or epochs) required for the test accuracy to reach 100%. If it hits a prolonged plateau (grokking), let it run until it generalizes.
- Print out the final "Energy Cost" (total steps to perfect generalization) so it can be compared against future experimental methods.

Please ensure the code is clean, heavily commented, and explicitly highlights the section where new optimization update rules can be injected.